Bayesian graph convolutional neural networks

Mark Coates[†]

Collaborators: Soumyasundar Pal[†], Yingxue Zhang[*], Deniz Üstebay[*]

[†]McGill University, [*]Huawei Noah's Ark Lab

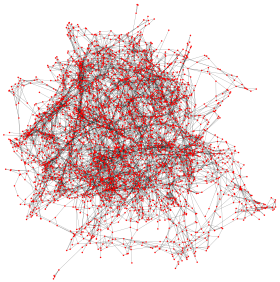
February 13, 2019

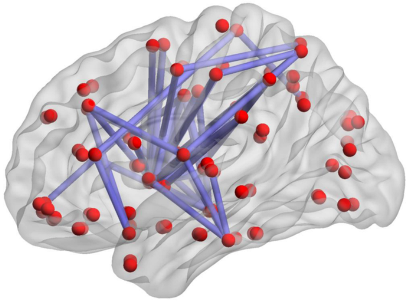
Montreal

ROBERT-BOURASSA / STE-CATHERINE
367-316 *
Banque Scotia Express
Sainte-Catherine
CAT
13.Feb 2019 08:05:06

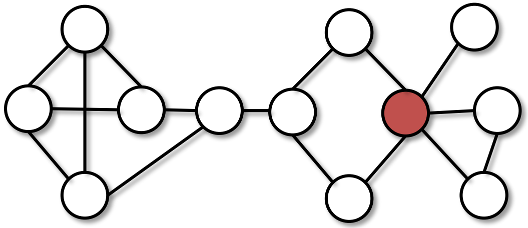
Introduction

- Exploit underlying graph structure to improve learning
- Many applications: cellular network configuration; molecular and social network analysis
- Focus on semi-supervised learning


Wireless Cellular Network

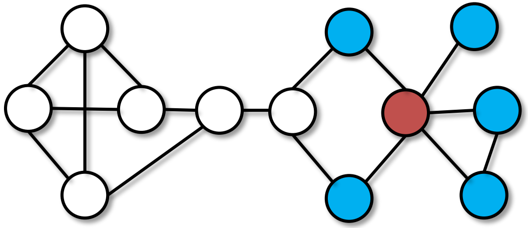

Brain Functional Connectivity
Reproduced from Hong S-B et al. (2013), Plos ONE 8(2):e57831.

Problem Setting



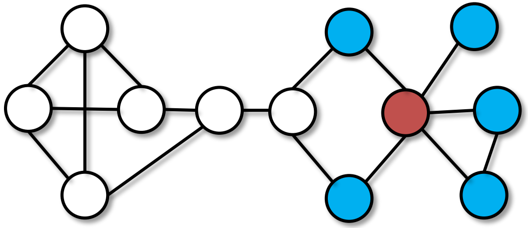
- Features available at each node $\mathbf{x}_i, i = 1, \ldots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 1: Ignore graph, learn function $\hat{y}_i = \hat{f}(\mathbf{x}_i)$

Problem Setting



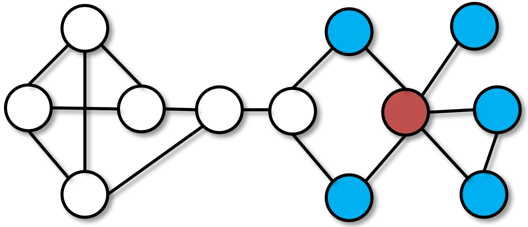
- Features available at each node $\mathbf{x}_i, i = 1, \dots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 2: Use graph, learn function $\hat{y}_i = \hat{f}(\mathbf{x}_i, \{\mathbf{x}_j\}_{j \in \mathcal{N}_i})$

Problem Setting



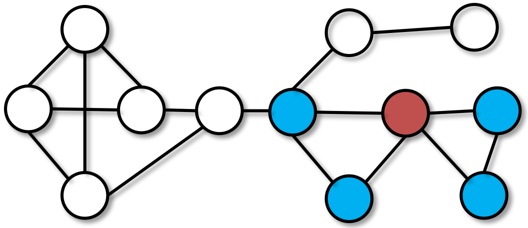
- Features available at each node $\mathbf{x}_i, i = 1, \ldots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 2: Use graph, learn function $\hat{y}_i = \hat{f}_{\mathcal{G}}(\mathbf{x}_i, \{\mathbf{x}_j\}_{j \in \mathcal{N}_i})$

What if we don't believe in the graph?



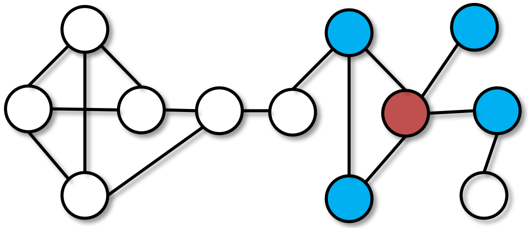
- Features available at each node $\mathbf{x}_i, i = 1, \ldots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 3: ?

What if we don't believe in the graph?



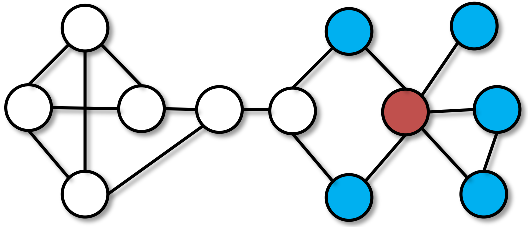
- Features available at each node $\mathbf{x}_i, i = 1, \ldots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 3: ?

What if we don't believe in the graph?



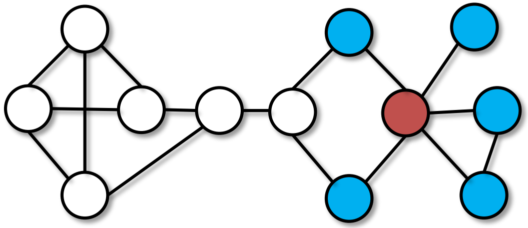
- Features available at each node $\mathbf{x}_i, i = 1, \ldots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 3: ?

What if we don't believe in the graph?



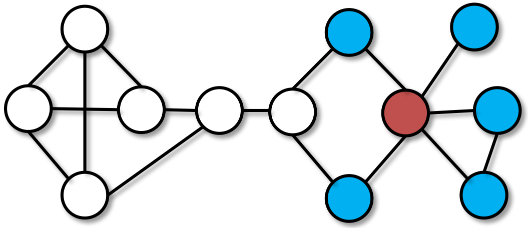
- Features available at each node $\mathbf{x}_i, i = 1, \ldots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 3: ?

What if we don't believe in the graph?



- Features available at each node $\mathbf{x}_i, i = 1, \ldots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 3: $\hat{y}_i = \int \hat{f}_{\mathcal{G}}(\mathbf{x}_i, \{\mathbf{x}_j\}_{j \in \mathcal{N}_{i,\mathcal{G}}}) p(\mathcal{G}|\mathcal{G}_{obs}) d\mathcal{G}$

What if we don't believe in the graph?



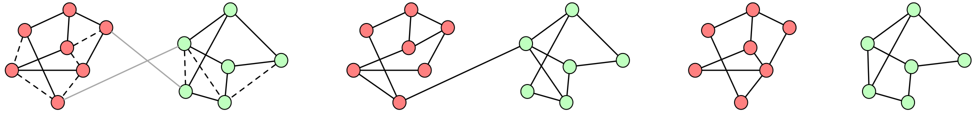
- Features available at each node $\mathbf{x}_i, i = 1, \ldots, N$
- Labels available at some nodes $y_i, i \in \mathcal{Y}_T$
- Approach 3: $\hat{y}_i = \frac{1}{K} \sum_{v=1}^{K} \hat{f}_{\mathcal{G}_v}(\mathbf{x}_i, \{\mathbf{x}_j\}_{j \in \mathcal{N}_{i,\mathcal{G}}})$

Background and motivation

- Graph Convolutional Neural Networks (GCNNs) use convolution on the graph
- In existing methods, the observed graph $\mathcal{G}_{obs}$ is processed as ground truth

- The graph is often derived from imperfect observations or constructed from noisy data
- $\mathcal{G}_{obs}$ might have spurious links; important links might not have been observed

- Our contribution: Bayesian framework to account for graph uncertainty



Graph Convolutional Neural Networks (GCNNs)

Graph convolutional layer[1] with adjacency matrix A and node feature matrix $\boldsymbol{X}$:

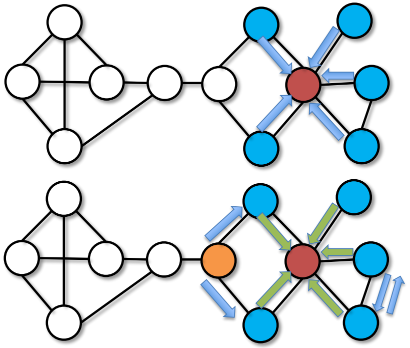
$$\mathbf{H}^{(1)} = \sigma(\mathbf{A}_{\mathcal{G}}\mathbf{X}\mathbf{W}^{(0)}) \tag{1}$$

$$\mathbf{H}^{(\ell+1)} = \sigma(\mathbf{A}_{\mathcal{G}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}) \tag{2}$$

$\mathbf{A}_{\mathcal{G}}$: operator derived from the adjacency matrix

$\mathbf{W}^{(\ell)}$: weights of neural network at layer ℓ

$\mathbf{H}^{(\ell)}$: output features from layer $\ell - 1$



[1] Defferrard et al. 2016; Kipf and Welling 2017

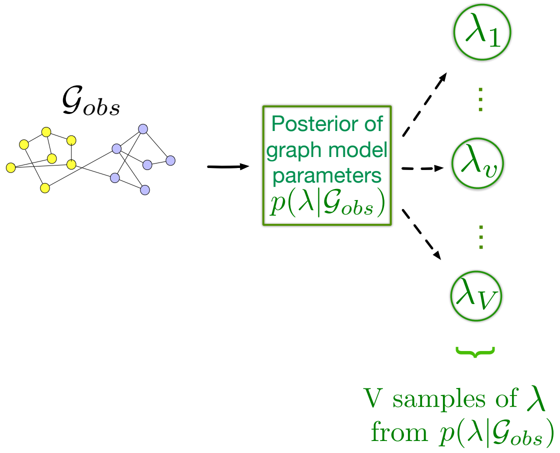
Bayesian-GCNNs

- In Bayesian neural networks[2], weights W are treated as random variables.
- Posterior of W is approximated via variational inference or sampling.
- Bayesian GCNN treats both the graph $\mathcal{G}$ and the weights W as random variables.
- Goal: Given node features $\mathbf{X}$, training labels $\mathbf{Y}_{\mathcal{L}}$, and an observed graph $\mathcal{G}_{obs}$:

$$\boxed{\text{Compute/approximate the posterior of the node labels: } p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs})}$$

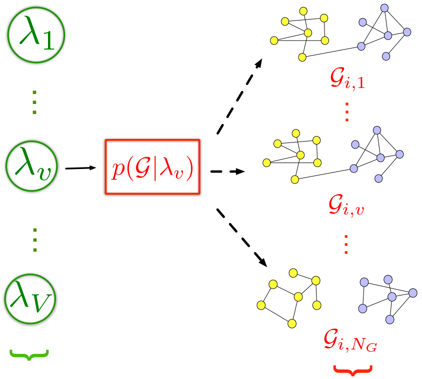
[2] Tishby et al. 1989; Denker and Lecun 1991; MacKay 1992; Neal 1993; Gal and Ghahramani 2016

Bayesian inference for a graph generative model


$\mathcal{G}_{obs}$
Posterior of graph model parameters $p(\lambda|\mathcal{G}_{obs})$
λ_1
λ_v
λ_V
V samples of λ from $p(\lambda|\mathcal{G}_{obs})$

$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}},\mathbf{X},\mathcal{G}_{obs}) = \int p(\mathbf{Z}|W,\mathcal{G},\mathbf{X})p(W|\mathbf{Y}_{\mathcal{L}},\mathbf{X},\mathcal{G})p(\mathcal{G}|\lambda)p(\lambda|\mathcal{G}_{obs})\,dW\,d\mathcal{G}\,d\lambda\,.$$

Sampling random graphs

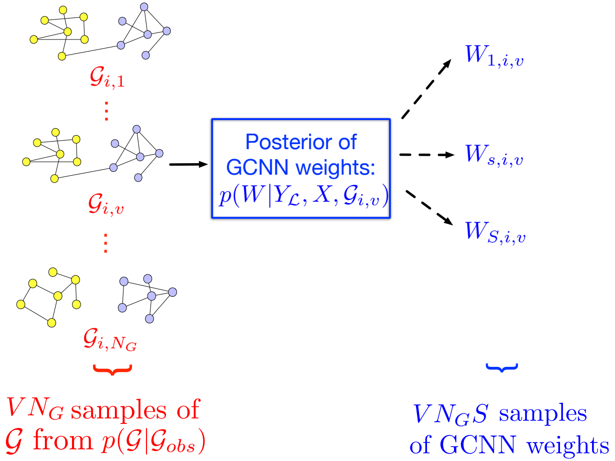


V samples of λ from $p(\lambda|\mathcal{G}_{obs})$

VN_G samples of $\mathcal{G}$ from $p(\mathcal{G}|\mathcal{G}_{obs})$

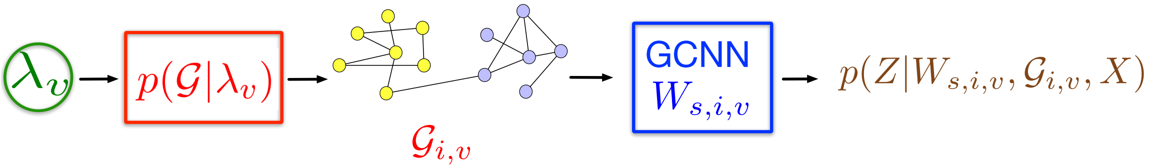
$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}},\mathbf{X},\mathcal{G}_{obs}) = \int p(\mathbf{Z}|W,\mathcal{G},\mathbf{X})p(W|\mathbf{Y}_{\mathcal{L}},\mathbf{X},\mathcal{G})p(\mathcal{G}|\lambda)p(\lambda|\mathcal{G}_{obs})\,dW\,d\mathcal{G}\,d\lambda\,.$$

Sampling GCNN weights


$\mathcal{G}_{i,1}$
$\mathcal{G}_{i,v}$
$\mathcal{G}_{i,N_G}$
Posterior of GCNN weights: $p(W|Y_{\mathcal{L}}, X, \mathcal{G}_{i,v})$
$W_{1,i,v}$
$W_{s,i,v}$
$W_{S,i,v}$
VN_G samples of $\mathcal{G}$ from $p(\mathcal{G}|\mathcal{G}_{obs})$
VN_GS samples of GCNN weights

$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs}) = \int p(\mathbf{Z}|W, \mathcal{G}, \mathbf{X}) p(W|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}) p(\mathcal{G}|\lambda) p(\lambda|\mathcal{G}_{obs})\, dW\, d\mathcal{G}\, d\lambda\,.$$

Computing the posterior of the node labels

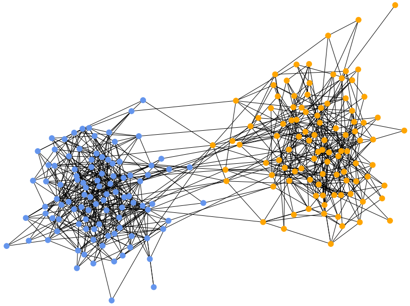

λ_v
$p(\mathcal{G}|\lambda_v)$
$\mathcal{G}_{i,v}$
GCNN
$W_{s,i,v}$
$p(Z|W_{s,i,v},\mathcal{G}_{i,v},X)$

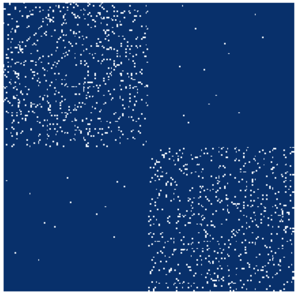
$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}},\mathbf{X},\mathcal{G}_{obs}) = \int p(\mathbf{Z}|W,\mathcal{G},\mathbf{X})p(W|\mathbf{Y}_{\mathcal{L}},\mathbf{X},\mathcal{G})p(\mathcal{G}|\lambda)p(\lambda|\mathcal{G}_{obs})\,dW\,d\mathcal{G}\,d\lambda\,,$$

$$\approx \frac{1}{V}\sum_{v=1}^{V}\frac{1}{N_G S}\sum_{i=1}^{N_G}\sum_{s=1}^{S} p(\mathbf{Z}|W_{s,i,v},\mathcal{G}_{i,v},\mathbf{X})\,.$$

Implementation details

- Assortative Mixed Membership Stochastic Block Model (MMSBM)[3] as $p(\mathcal{G}|\lambda)$
- Stochastic gradient-based MAP estimation
- Monte Carlo (MC) dropout[4] for sampling W





[3] Li, Ahn, and Welling 2016

[4] Gal and Ghahramani 2016

Aside: Bayesian neural networks

- Place prior: $p(\mathbf{W}_i)$ on weights of neural L-layer network

$$\mathbf{W}_i \sim \mathcal{N}(0, \mathbf{I})$$

for $i \leq L$ (and write $\omega := \{\mathbf{W}_i\}_{i=1}^{L}$)

Aside: Bayesian neural networks

- Place prior: $p(\mathbf{W}_i)$ on weights of neural L-layer network

$$\mathbf{W}_i \sim \mathcal{N}(0, \mathbf{I})$$

  for $i \leq L$ (and write $\omega := \{\mathbf{W}_i\}_{i=1}^{L}$)
- Output is a random variable

$$f(\mathbf{x}, \omega) = \mathbf{W}_L \sigma(\dots \mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \dots)$$

- Softmax likelihood for classification: $p(y|\mathbf{x}, \omega) = \text{softmax}(f(\mathbf{x}, \omega))$ or a Gaussian for regression: $p(\mathbf{y}|\mathbf{x}, \omega) = \mathcal{N}(\mathbf{y}; f(\mathbf{x}, \omega), \tau^{-1}\mathbf{I})$

Aside: Bayesian neural networks

- Place prior: $p(\mathbf{W}_i)$ on weights of neural L-layer network

$$\mathbf{W}_i \sim \mathcal{N}(0, \mathbf{I})$$

  for $i \leq L$ (and write $\omega := \{\mathbf{W}_i\}_{i=1}^{L}$)
- Output is a random variable

$$f(\mathbf{x}, \omega) = \mathbf{W}_L \sigma(\ldots \mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \ldots)$$

- Softmax likelihood for classification: $p(y|\mathbf{x}, \omega) = \text{softmax}(f(\mathbf{x}, \omega))$ or a Gaussian for regression: $p(\mathbf{y}|\mathbf{x}, \omega) = \mathcal{N}(\mathbf{y}; f(\mathbf{x}, \omega), \tau^{-1}\mathbf{I})$
- Very difficult to evaluate the posterior: $p(\omega|\mathbf{x}, \mathbf{y})$

Approximate inference in Bayesian neural networks

- Define $q_\theta(\omega)$ to approximate the posterior $p(\omega|\mathbf{x}, \mathbf{y})$
- Minimize KL divergence:

$$
\begin{aligned}
&KL(q_\theta(\omega)||p(\omega|\mathbf{x}, \mathbf{y}) \\
&\quad \propto - \int q_\theta(\omega) \log p(\omega|\mathbf{x}, \mathbf{y}) d\omega + KL(q_\theta(\omega)||p(w)) \\
&\quad =: \mathcal{L}(\theta)
\end{aligned}
$$

- Approximate the integral with MC integration $\hat{\omega} \sim q_\theta(\omega)$:

$$\hat{\mathcal{L}}(\theta) = - \log p(\mathbf{y}|\mathbf{x}, \hat{\omega}) + KL(q_\theta(\omega)||p(w))$$

Stochastic inference in Bayesian neural networks

- Unbiased estimator:

$$\mathbf{E}_{\hat{\omega}\sim q(\omega)}(\hat{\mathcal{L}}(\theta)) = \mathcal{L}(\theta)$$

- Converges to the same optima as $\mathcal{L}(\theta)$
- For inference, repeat:
  1. Sample $\hat{\omega} \sim q_\theta(\omega)$.
  2. Minimise (one step) w.r.t. θ

$$\mathcal{L}(\theta) = \log p(\mathbf{y}|\mathbf{x}, \hat{\omega}) + KL(q_\theta(\omega)||p(\omega))$$

Stochastic inference in Bayesian neural networks

- Need to specify $q_\theta(\cdot)$:
- Given $z_{i,j}$ Bernoulli random variables
- Variational parameters $\theta = \{\mathbf{M}_i\}_{i=1}^{L}$ (set of matrices):

$$
\begin{aligned}
z_{i,j} &\sim \mathit{Bernoulli}(p_i) \text{ for } i = 1, \ldots, L, \; j = 1, \ldots, K_{i-1} \\
\mathbf{W}_i &= \mathbf{M}_i \cdot \mathrm{diag}([z_{i,j}]_{j=1}^{K_i}) \\
q_\theta(\omega) &= q_{\mathbf{M}_i}(\mathbf{W}_i)
\end{aligned}
$$

Stochastic inference in Bayesian neural networks

- Repeat:
  1. Sample $\hat{z}_{i,j} \sim \text{Bernoulli}(p_i)$ and set:

$$\hat{\mathbf{W}}_i = \mathbf{M}_i \cdot \text{diag}([\hat{z}_{i,j}]_{j=1}^{K_i})$$
$$\hat{\omega} = \{\hat{\mathbf{W}}_i\}_{i=1}^{L}$$

  2. Minimise (one step) w.r.t. $\theta = \{\mathbf{M}_i\}_{i=1}^{L}$

$$\mathcal{L}(\theta) = \log p(\mathbf{y}|\mathbf{x}, \hat{\omega}) + KL(q_\theta(\omega)||p(\omega))$$

Stochastic inference in Bayesian neural networks

- Repeat:
  1. Randomly set columns of $\mathbf{M}_i$ to zero
  2. Minimise (one step) w.r.t. $\theta = \{\mathbf{M}_i\}_{i=1}^{L}$

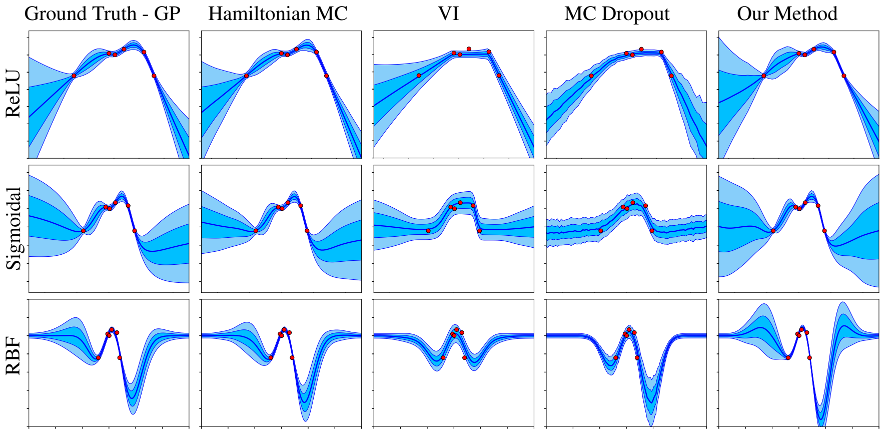
$$\mathcal{L}(\theta) = \log p(\mathbf{y}|\mathbf{x}, \hat{\omega}) + KL(q_\theta(\omega)||p(\omega))$$

Stochastic inference in Bayesian neural networks

- Repeat:
  1. Randomly set units of the network to zero $\Rightarrow$ Dropout
  2. Minimise (one step) w.r.t. $\theta = \{\mathbf{M}_i\}_{i=1}^{L}$

$$\mathcal{L}(\theta) = \log p(\mathbf{y}|\mathbf{x}, \hat{\omega}) + KL(q_\theta(\omega)||p(\omega))$$

Are we really sampling from the posterior?

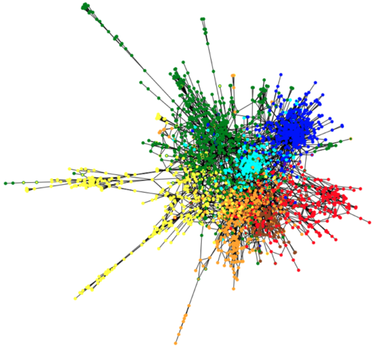


- T. Pearce, M. Zaki and A. Neely, “Bayesian Neural Network Ensembles”, Proc. Workshop on Bayesian Deep Learning (NeurIPS 2018), Montral, Canada.

Experimental results: Citation network classification

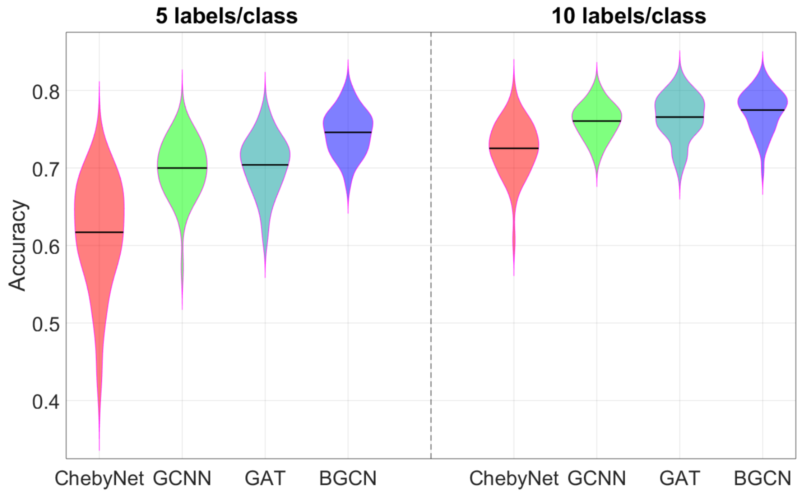
	Cora	**CiteSeer**	**Pubmed**
Nodes	2708	3327	19717
Edges	5429	4732	44338
Features per node	1433	3703	500
Classes	7	6	3

- 5/10/20 training examples per class
- Random splitting of training and test data
- 50 trials per experiment setting
- Comparison with ChebyNet[5], GCNN[6], and GAT[7]



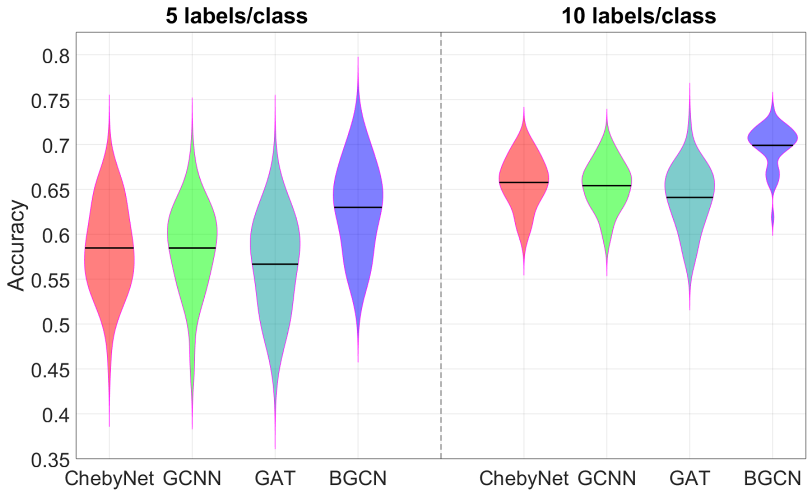
Sen et al. 2008; 5: Defferrard et al. 2016; 6: Kipf & Welling 2017; 7: Veličković et al. 2018

Semi-supervised node classification for Cora



When training data is limited, BGCN outperforms competing techniques

Semi-supervised node classification for Citeseer



When training data is limited, BGCN outperforms competing techniques

Node classification under graph attacks

- Goal: Examine robustness to corruption or attack[8]
- For node v with true class c_{true}, classification margin is:

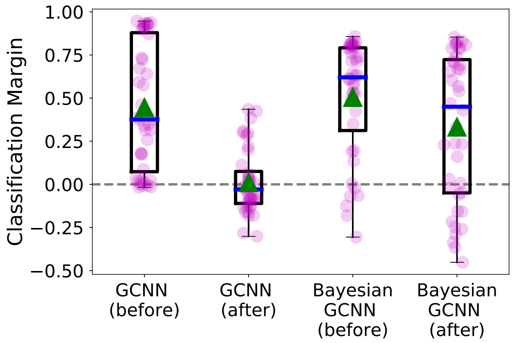
$$\text{margin}_v = \text{score}_v(c_{true}) - \max_{c \neq c_{true}} \text{score}_v(c)\,.$$

- Select 40 nodes for attack, based on classification margin.
- Node perturbation: $\Delta = d_v + 2$, where d_v is the degree of node v
- Remove $\frac{\Delta}{2}$ random edges; add $\frac{\Delta}{2}$ cross-community edges

[8] Zügner, Akbarnejad and Günnemann 2018

Node classification under graph attacks

	No attack	Random attack
	Accuracy	
GCNN	88.5%	43.0%
Bayesian GCNN	87.0%	66.5%
	Classifier margin	
GCNN	0.448	0.014
Bayesian GCNN	0.507	0.335



Conclusion

- Compared to existing algorithms, Bayesian-GCNNs have :
  - better performance with limited training data
  - more resilience to random perturbations of the graph topology
  - principled methodology to represent uncertainty
- The general Bayesian framework can incorporate:
  - a variety of generative models for graphs
  - different inference techniques for the graph generative model
  - different versions of graph based learning algorithms